Find Merge Point of Two Lists locked

This challenge is part of a tutorial track by MyCodeSchool

Given pointers to the head nodes of **2** linked lists that merge together at some point, find the node where the two lists merge. The merge point is where both lists point to the same node, i.e. they reference the same memory location. It is guaranteed that the two head nodes will be different, and neither will be NULL. If the lists share a common node, return that node's ***data*** value.

Note: After the merge point, both lists will share the same node pointers.

Example

In the diagram below, the two lists converge at Node `x`:

```
[List #1] a--->b--->c
                    \
                     x--->y--->z--->NULL
                    /
     [List #2] p--->q
```

Function Description

Complete the *findMergeNode* function in the editor below.

findMergeNode has the following parameters:

- *SinglyLinkedListNode pointer head1:* a reference to the head of the first list
- *SinglyLinkedListNode pointer head2:* a reference to the head of the second list

Returns

- *int:* the ***data*** value of the node where the lists merge

Input Format

Do not read any input from stdin/console.

The first line contains an integer t, the number of test cases.

Each of the test cases is in the following format:
The first line contains an integer, $index$, the node number where the merge will occur.
The next line contains an integer, $list1_count$ that is the number of nodes in the first list.
Each of the following $list1_count$ lines contains a $data$ value for a node. The next line contains an integer, $list2_count$ that is the number of nodes in the second list.
Each of the following $list2_count$ lines contains a $data$ value for a node.

Constraints

The lists will merge.
$head1, head2 \neq null$.
$head1 \neq head2$.

Sample Input

The diagrams below are graphical representations of the lists that input nodes $head1$ and $head2$ are connected to.

Test Case 0

```
1
 \
  2--->3--->NULL
 /
1
```

Test Case 1

```
1--->2
      \
       3--->Null
      /
     1
```

Sample Output

```
2
3
```

Explanation

Test Case 0: As demonstrated in the diagram above, the merge node's data field contains the integer 2.
Test Case 1: As demonstrated in the diagram above, the merge node's data field contains the integer 3.

Submissions: 120
Max Score: 30
Difficulty: Easy

Rate This Challenge:

More

C++14

```
#include ⇔

using namespace std;

class SinglyLinkedListNode {
    public:
        int data;
        SinglyLinkedListNode *next;

        SinglyLinkedListNode(int node_data) {
            this->data = node_data;
            this->next = nullptr;
        }
};

class SinglyLinkedList {
    public:
        SinglyLinkedListNode *head;
        SinglyLinkedListNode *tail;

        SinglyLinkedList() {
            this->head = nullptr;
            this->tail = nullptr;
        }
```

```
        void insert_node(int node_data) {
            SinglyLinkedListNode* node = new SinglyLinkedListNode(node_data);

            if (!this->head) {
                this->head = node;
            } else {
                this->tail->next = node;
            }

            this->tail = node;
        }
};

void print_singly_linked_list(SinglyLinkedListNode* node, string sep, ofstream& fout) {
    while (node) {
        fout << node->data;

        node = node->next;

        if (node) {
            fout << sep;
        }
    }
}

void free_singly_linked_list(SinglyLinkedListNode* node) {
    while (node) {
        SinglyLinkedListNode* temp = node;
        node = node->next;

        free(temp);
    }
}
```

```
// Complete the findMergeNode function below.

/*
 * For your reference:
 *
 * SinglyLinkedListNode {
 *     int data;
 *     SinglyLinkedListNode* next;
 * };
 *
 */
int findMergeNode(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2) {


}
```

```
int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    int tests;
    cin >> tests;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    for (int tests_itr = 0; tests_itr < tests; tests_itr++) {
        int index;
        cin >> index;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        SinglyLinkedList* llist1 = new SinglyLinkedList();

        int llist1_count;
        cin >> llist1_count;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        for (int i = 0; i < llist1_count; i++) {
            int llist1_item;
```

```
            cin >> llist1_item;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            llist1->insert_node(llist1_item);
        }

        SinglyLinkedList* llist2 = new SinglyLinkedList();

        int llist2_count;
        cin >> llist2_count;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        for (int i = 0; i < llist2_count; i++) {
            int llist2_item;
            cin >> llist2_item;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            llist2->insert_node(llist2_item);
        }

        SinglyLinkedListNode* ptr1 = llist1->head;
        SinglyLinkedListNode* ptr2 = llist2->head;

        for (int i = 0; i < llist1_count; i++) {
            if (i < index) {
                ptr1 = ptr1->next;
            }
        }

        for (int i = 0; i < llist2_count; i++) {
            if (i != llist2_count-1) {
                ptr2 = ptr2->next;
            }
        }

        ptr2->next = ptr1;

        int result = findMergeNode(llist1->head, llist2->head);

        fout << result << "\n";
    }

    fout.close();

    return 0;
}
```

Line: 17 Col: 1

Upload Code as File Test against custom input Run Code Submit Code